NTNU
Det skapende universitet

Datasikkerhet i sikkerhetsinstrumenterte systemer

Thea Kristine Thorrud

Master i kybernetikk og robotikk
Innlevert: juni 2016
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Forord

Denne masteroppgaven er skrevet for Institutt for Teknisk kybernetikk på Norges teknisk-naturvitenskapelige universitet (NTNU) våren 2016, som det avsluttende arbeidet på min mastergrad (M.Sc.) innen Teknisk Kybernetikk.

Under arbeidet med denne oppgaven har jeg lært utrolig mye nytt, både relevant og ikke relevant. Uansett er dette kunnskap jeg tar med meg videre og kommer til å ha glede av. Jeg har også fått lov til å fordype meg i et fagfelt jeg synes er interessant, og som jeg nå føler jeg har mye kunnskap om, selv om jeg hadde veldig lite forkunnskaper da jeg begynte på denne oppgaven.

Jeg ønsker å takke følgende personer:

- Min veileder, Professor Tor Onshus. Takk for hjelp og veiledning, og for at du lot meg fordype meg i et emne jeg ikke kunne så mye om. Jeg har lært desto mer.

- Alle jeg har delt kontor G-238 med det siste året. Arbeidsdagen hadde ikke vært den samme uten dere.

Trondheim, 2016-06-06

Thea Kristine Thorrud

thea.k.thorrud@gmail.com

Sammendrag og Konklusjon

Ved oppstarten av denne oppgaven hadde forfatteren ingen forkunnskaper om kryptering eller datasikkerhet, og kun begrenset kjennskap til sikkerhetsinstrumenterte systemer (SIS).

Instrumenteringssystemer har tradisjonelt vært separate systemer og har krevd lite fokus på datasikkerhet. I senere tid har økende bruk av standard kommunikasjonsprotokoller, kommunikasjon over lange avstander, samt utstyr koblet til internett, gjort kommunikasjon med instrumenteringssystemer betydelig mer utsatt for nettverksangrep enn tidligere. Mange bedrifter begynner å se betydelige fordeler ved å satse på IO (Integrerte Operasjoner), noe som har vært en stor pådriver for denne endringen. Spesielt i oljebransjen vil IO kunne resultere i vesentlige kostnadsbesparelser og redusert risiko, da mange operasjoner kan flyttes fra plattformen og inn til land.

Med innføringen av IO kommer også et økt behov for datasikkerhet i instrumenteringssystemer. Samtidig er det de samme instrumenteringssystemene som styrer sikkerhetskritiske funksjoner som PAS (Prosess Avstengnings Systemer) og NAS (Nød Avstengnings Systemer). Det er et krav at eventuelle tiltak for datasikkerhet (security) ikke kommer på bekostning av kontrollsystemets sikkerhetsfunksjoner (safety). Formålet med denne oppgaven er å kartlegge hvilke krav som stilles til kommunikasjon mellom kontrollrom og sikkerhetskritiske styringsenheter i felt, og hvordan dette kan kombineres med tiltak for å sikre kommunikasjonen mot nettverksangrep.

For å utforske fremgangsmåter og utfordringer ble det gjort et forsøk på å opprette en sikker kommunikasjonskanal mellom en styringsenhet (representert av en Arduino Mega 2560) og en PC. Som kommunikasjonsmedie ble det benyttet ethernet med en TCP/IP tilkobling.

Et av de viktigste tiltakene for sikre kommunikasjon mot innblanding utenfra, er å kryptere informasjonen som blir sendt. Det finnes et stort antall krypteringsalgoritmer, men mange er allerede “knekt”, det finnes altså kjente prosedyrer for å hente ut det krypterte innholdet. Dette vil i praksis gjøre krypteringen ubrukelig, og i mange tilfeller være verre enn å ikke bruke kryptering i det hele tatt, da det vil gi en falsk trygghet. Ettersom den tilgjengelige mengden regnekraft og lagringskapasitet øker, vil stadig flere krypteringsalgoritmer bli knekt. Det er derfor ekstremt viktig at de som drifter systemene holder seg konstant oppdatert på nye utviklinger og sørger for at de algoritmene som er i bruk fremdeles er sikre. Kryptering som benyttes i kontrollsystemer bør også være rask og effektiv, ettersom det gjerne er tidsbegrensninger i et kontrollsystem. Det som blir kryptert er gjerne korte meldinger

som sendes som en datapakke. For å spare båndbredde er det derfor også ønskelig å benytte krypteringsalgoritmer med så lite "overhead" (tilleggsinformasjon som må legges ved den krypterte meldingen) som mulig.

I denne oppgaven er det kommet frem til at krypteringsalgoritmen AES med operasjonsmodus EAX er et fornuftig valg for kryptering i et kontrollsystem, da denne kombinasjonen har flere viktige kvaliteter. Blant disse kan det nevnes at EAX er en effektiv og rask kryptering, laget for å erstatte en lignende men tregere og mer kompleks motpart: CCM. Både AES og EAX bidrar til lite "overhead". EAX kan også benyttes til kryptering av meldinger med ulik lengde. Dette fjerner behovet for "padding", å utvide en melding med nuller eller tilfeldige tall for at lengden skal passe inn i krypteringsalgoritmen. Det er ikke ønskelig å benytte mye padding ved kryptering av korte meldinger, ettersom det vil gjøre krypteringen tregere, og meldingene som skal sendes lengre enn nødvendig.

En av de største utfordringene med datasikkerhet i et instrumenteringssystem er at systemene ofte ikke kan tolerere nedetid for ulike utskiftninger og software oppdateringer. Som et eksempel vil forsvarlig nedstengning av et oljeraffineri fort ta mange dager, om ikke uker. Alle utskiftinger og oppdateringer må derfor kunne utføres mens systemet er "online" eller kunne utsettes til systemet har planlagt nedetid. Modularitet er derfor viktig, ettersom det gjør det enklere å gjøre endringer uten at det fører til nedetid.

Ulike nettverksstrukturer er et viktig virkemiddel for å beskytte SIS. Brannmurer og Demilitariserte soner (DMZ) kan brukes til å dele opp et nettverk i ulike lag, og gjøre det vanskelig for angripere å få tilgang til sikkerhetskritiske funksjoner.

I SIS er en også avhengig av at feil i et system vil få det til å feile på en sikker måte (En. failsafe) og gå til en forhåndsdefinert sikker tilstand. Dette vil ha ulike betydninger i forskjellige tilfeller, men vil i mange tilfeller innebære avstenging av en prosess, ved hjelp av et NAS eller PAS. Denne ideen om å feile sikkert kan overføres til datasikkerhet. Et angrep på et SIS bør aldri kunne nedsette systemets evne til å feile på en sikker måte. En konsekvens av dette er at et oppdaget nettverksangrep med potensiale til å påvirke SIS bør føre til at systemet går til sikker tilstand. Det vil også være viktigere å oppdage et angrep for så å gå til sikker tilstand, enn det er å forsøke å hindre angrepet fra å skje. Det er ingen måter man kan sørge for å være fullstendig beskyttet mot angrep, men det finnes måter å sikre at et angrep fører til minst mulig skade.

Summary and conclusion

At the start of this thesis, the writer had no previous knowledge about encryption or computer-security, and only limited knowledge about safety instrumented systems (SIS)

Instrumented systems has traditionally been separate systems and has required little focus on computer-security. In recent times increasing use of standard communication-protocols, communication over vast distances and equipment connected to the internet has made communication with instrumentation-systems significantly more exposed to network-attacks than before. Many companies have started to see significant benefits to investing in IO (Integrated Operations), something that has been an important contributor to this change. Especially within the offshore oil and gas industry, IO could lead to significantly reduced cost and risk, as many people and operations can be moved onshore.

The widened use of IO also brings an increased need for computer-security in instrumented systems. At the same time, it is the same instrumented systems that control critical safety-functions like emergency shutdown systems. It is an absolute requirement that no measures taken to ensure computer-security will come at the cost of the systems safety. The purpose of this thesis is to map which requirements exist in communication between process control systems and safety-critical control units in the field, and how this can be combined with computer-security.

To explore different challenges and approaches to this, it was made an attempt to create a secure communicationchannel between a control-unit (represented by an Arduino Mega 2560) and a PC. Ethernet with a TCP/IP connection was used for this purpose.

one of the most important measures to secure communication from tampering from the outside is to encrypt all infirmation that is sent. There is a large number of encryption-algorithms to choose from, but many are already “broken”, meaning that there exists procedures for outsiders to extract the unencrypted contents of an encrypted text. If such an algorithm is used, it will be practically useless, and in many cases worse than having no encryption at all, as it will give a false sense of security. As the accessible amount of computing power and storagecapacity increases, more algorithms will be broken. It is therefore extremely important to that the people responsible for instrumented systems keep themselves updatet on new developments and makes sure that the currently used algorithms are still secure. Encryption in control-systems should be fast and efficient, as there usually are some timing concerns in such systems. The text that gets encrypted are usually small packets of data. To

save time and bandwith it is desirable to use an encryptionalgorithm that produces as little overhead as possible.

In this thesis it has been argued that the encryptionalgorithm “AES” (Advanced Encryption Standard) with the “EAX” mode of operation is a sensible choice for encryption in a control-system, as this combination has several desired qualities. EAX is fast and efficient, made to replace a similar but slower and more complex counterpart: CCM. Both AES and EAX produce little overhead. EAX can be sed for encrytion of messages with different lengths. This removes the need for padding (to expand a message to fit a certain size requirement for encrytion). It is not desirable to use a lot of padding in encryption of short messages, s it will make the encryption slower and the messages to be sent longer than strictly necessary.

One of the greatest challenges with adding computer-security to instrumented systems is that the systems often cannot tolerate downtime. Examples of this is atomic reactors and oil refineries. Safely shutting down an oil refinery can take days, if not weeks. All changes and updates must be done while the system is “online”, and it must be done without risk. Modularity i therefore of the highest importance, as it simplifies making changes without downtime.

Different networkstructures are an important tool for protecting a SIS. Firewalls and demilitarized zones (DMZ) can be used to split up a network into different layers, to make it hard for an attacker to gain access to SIS.

In a SIS it is also necessary that an error in a system will make it fail in a safe way, a failsafe, and go to a previously defined safe state. This will have different meanings in different systems, but will in many cases include the activation of the emergency shutdown procedure. This idea, to fail safe, kan be carried on to computer-security. An attack on a SIS should never compromise the systems ability to fail safely. A consquence of this, is that a discivered attack with potential to affect the SIS, should lead to the system failsafe state being activated. It will also be more important to discover an attack and activate the failsafe state, than to try an hinder the attack from occuring. There is no way to be completely safe from attaks, but there is a way to ensure an attack causes the minimal amount of damage.

Liste over forkortelser og definisjoner

AES Advanced Encryption Standard (Avansert krypteringsstandard)

CRC Cyclic Redundancy Check (Syklisk redundant sjekk)

DES Data Encryption Standard (Data krypteringsstandard)

DMZ Demilitarized Zones (Demilitariserte soner)

DoS Denial of Service

EAX Operasjonsmodus for blokk-kryptering. Det er usikkert hva EAX egentlig står for.

EFD End of Frame Delimiter (Spesiell bitsekvens som markerer slutten av en ethernet-melding)

FCS Frame Checking Sequence (Meldingssjekk sekvens)

GCM Galois/Counter Mode (Operasjonsmodus for blokk-kryptering)

HMS Helse, Miljø og Sikkerhet

IO Integrerte Operasjoner

IV Initialiserings Variabel

LED Light Emitting Diode (Lysdiode)

MAC Message Autentication Code [kode] (Meldings-Autentiserings-Kode)

MAC Medium Access Control [adresse] (Fysisk internett adresse)

NAS Nød Avstengings System

NIST National Institute of Standards and Technology

OCB Offset CodeBook Mode (Operasjonsmodus for blokk-kryptering)

PAS Prosess Avstengnings System

PCS Process Control System (Prosess kontroll system)

RSA Et offentlig-nøkkel kryptografisystem patentert av Rivest, Shamir og Adelmann

SAS Safety and Automation System (Sikkehets og Automasjons System)

SFD Start of Frame Delimiter (Spesiell bitsekvens som markerer starten av en ethernet-melding)

SIS Sikkerhets Instrumenterte Systemer

SPN Substitution-Permutation Networks (Substitusjon og permutasjons nettverk)

TCP Transmission Control Protocol (Overordnet protokoll for overføring av data over intenett)

UDP User Datagram Protocol (Overordnet protokoll for overføring av data over intenett)

Ordforklaringer

Dette er en liste over engelske ord og uttrykk som blir benyttet i rapporten.

Brute force: En form for kryptoanalyse som går ut på å teste alle mulige kombinasjoner av krypteringsnøkler. Meget ineffektiv alene.

Break: Et kryptografisk break er når det blir oppdaget metoder for å hente ut kryptert informasjon raskere enn med bruk av brute force.

Failsafe: Et systems evne til å feile på en sikker måte.

Handshake: Benyttes i flere kommunikasjonsprotokoller for å initialisere en tilkobling. Det inneholder forespørsler om tilkobling, og tilsvarende respons.

Header: Tilleggs informasjon som sendes med alle meldinger i en protokoll. Inneholder blant annet avsender/mottager-adresser og et felt som markerer starten på meldingen.

Chosen ciphertext attack: Et angrep der en angriper forsøker å få offeret til å dekryptere en valgt kryptert tekst for å få informasjon om krypteringsnøkkelen som er benyttet.

Chosen plaintext attack: Et angrep der en angriper forsøker å få offeret til å kryptere en valgt klartekst for å få informasjon om krypteringsnøkkelen som er benyttet.

Known ciphertext attack: I praksis det samme som brute force, en form for kryptoanalyse der angriper kun har tilgang til kryptert tekst.

Known plaintext attack: Et angrep der angriper er i besittelse av noe kryptert tekst med tilsvarende klartekst.

Cryptoanalysis: Kryptoanalyse: Å forsøke å dekryptere kryptert informasjon uten å kjenne til den benyttede krypteringsnøkkelen.

Padding: Nuller eller tilfeldige tall som legges til en tekst som skal krypteres for at den skal få en bestemt lengde.

Safety: Beskyttelse mot ulykker og utilsiktede handlinger som kan påvirke systemets sikkerhet.

Security: Beskyttelse mot tilsiktede og ondsinnede handlinger som kan påvirke systemets sikkerhet.

Innhold

Del I

Introduksjon

Kapittel 1

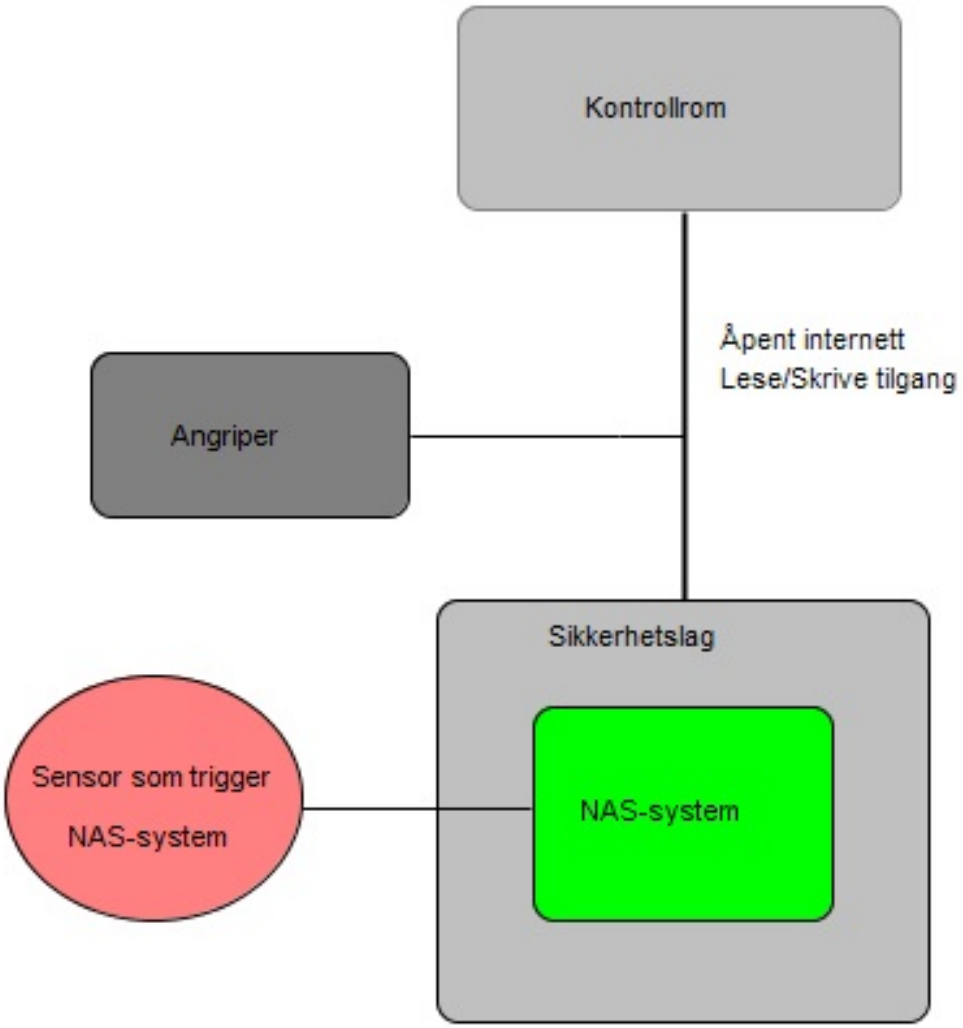
Introduksjon

1.1 Bakgrunn

Kontrollsystemer har lenge hatt et begrenset behov for datasikkerhet, da de gjerne har vært isolerte systemer som har kommunisert ved bruk av proprietære protokoller og programvare. Dette er ikke lenger tilfelle, og datasikkerhet er et stadig større felt innen kontrollsystemer. Økende bruk av standard kommunikasjonsprotokoller, kommunikasjon over lange avstander, samt utstyr koblet til internett, gjør kommunikasjon med kontrollsystemer betydelig mer utsatt for nettverksangrep. En drivende kraft bak bruken av internett innen kontrollsystemer er integrerte operasjoner (IO), hvor stadig flere kontrollfunksjoner sentraliseres i et kontrollrom langt unna det som skal kontrolleres. Dette er spesielt vanlig på oljeplattformer, der det kan spares mye på å flytte så mange funksjoner som mulig inn til kontrollrom på land. IO gir også redusert HMS-risiko (Helse, Miljø og Sikkerhet), da operatørene flyttes vekk fra plattformene og inn til land.

Det er spesielt viktig å beskytte de delene av et kontrollsystem som er kritiske for sikkerheten, de sikkerhetsinstrumenterte systemene (SIS). Samtidig er det et krav at eventuelle tiltak for datasikkerhet (security) ikke kommer på bekostning av kontrollsystemets sikkerhetsfunksjoner (safety).

Utgangspunktet for oppgaven er illustrert i figur 1.1. Et kontrollrom er koblet til et NAS (Nød Avstengnings System) via internett. Et sikkerhetslag rundt skal sørge for at kun er kontrollrommet som kan kommunisere med systemet, og ikke eventuelle angripere som er koblet til internett.



Figur 1.1: Generell skisse av oppgavens utgangspunkt

1.2 Mål for oppgaven

Formålet med denne oppgaven er å kartlegge hvilke krav som stilles til kommunikasjon mellom kontrollrom og sikkerhetskritiske styringsenheter i felt, og hvordan dette kan kombineres med tiltak for å sikre kommunikasjonen mot nettverksangrep. Som et ledd i denne prosessen vil det bli gjort forsøk på å implementere en sikker kommunikasjonskanal mellom en PC som fungerer som et PCS (Process Control System) og en styringsenhet (representert av en Arduino Mega 2560).

Mål for oppgaven kan sammenfattes i følgende punkter:

- Utrede hvilke sikkerhetstiltak som benyttes i dagens systemer.
- Forsøke å implementere sikker kommunikasjon over ethernet mellom en styringsenhet og et kontrollsystem.
- Evaluere behovene for ulike sikkerhetstiltak utifra utredning og implementasjon.

1.3 Fremgangsmåte

Det vil bli gjort forsøk på å sette opp sikker kommunikasjon mellom en PC som fungerer som et kontrollsystem, og en styringsenhet (en Arduino Mega 2560) med ethernet tilkobling. Kommunikasjonen skal være mest mulig robust, og sikres mot ulike nettverksangrep. For å gjennomføre dette vil det først bli gjort en utredning av hvilke typer nettverksangrep som er aktuelle og hvilke tiltak det er mulig å gjøre for å hindre slike angrep. Det vil også bli gjort en utredning av ulike utfordringer knyttet til å implementere de ulike tiltakene for datasikkerhet i sikkerhetskritiske kontrollsystemer.

1.4 Begrensninger

Grunnet tidsbegrensninger, er det flere tema som kunne vært inkludert i denne oppgaven, men som det ikke er tid til å utforske. Oppgaven vil i hovedsak fokusere på kommunikasjon, så det vil ikke være fokus på fysisk sikring av de kommuniserende enhetene. I tillegg til dette er det også flere aspekter ved oppgaven som kun vil bli diskutert teoretisk, ettersom det ikke er praktisk mulig å implementere disse med utstyret som er tilgjengelig. Dette gjelder blant

annet struktur i selve nettverket, brannmurer og autentisering av brukere. Det vil kun bli benyttet én styringsenhet i implementasjonen.

1.5 Strukturen i Rapporten

Del 1 er en kort innledning. Del 2 vil introdusere en del generell teori, samt en oppdatering over hvilke løsninger og problemer som er vanlige i dagens kontrollteknologi. Del 3 vil omhandle den fysiske kommunikasjonen som er satt opp, løsninger som er brukt i det fysiske oppsettet, samt begrunnelse for ulike designvalg gjort i prosessen. Del 4 vil inneholde en oppsummering av videre arbeid. Tekniske spesifikasjoner på utstyret som er brukt finnes i vedlegg A.

Del II

Teori

Kapittel 2

Sikkerhetstrusler

2.1 Nye utfordringer

Instrumenteringssystemer har tradisjonelt vært separate systemer og har krevd lite datasikkerhet. Endringer som har skjedd i måten kontrollsystemer implementeres har endret dette. Noen utfordringer er:

- Det ble tidligere brukt systemspesifikke kommunikasjonsprotokoller som gjør det vanskelig for utenforstående å gjøre endringer. Dette endres raskt, ettersom mange tar i bruk standard protokoller for kommunikasjon, slik som TCP/IP. Det gjør det lettere for utenforstående å ta seg inn i et system.

- Endring i måten ansatte jobber sammen. IT-løsninger trekkes i stadig større grad inn i kontrollsystemer, men kontrollingeniører er ikke eksperter på IT-løsninger, noe som krever større tverrfaglig samarbeid enn tidligere.

- Ønske om å kontrollere og overvåke prosesser på trygg (og økonomisk) avstand fører til at mye kontrollinformasjon passerer gjennom internett. Det åpner for sårbarheter, da (uautorisert) tilgang til bedriftens nettverk også kan gi direkte tilgang til kontrollsystemet.

- Oppdateringer og utskiftninger er etterhvert nødvendig, men omstart og midlertidig nedstenging av et kontrollsystem er i mange tilfeller ikke et alternativ (f. eks. oljeraffineri, kontroll for kraftnettet, kontrollenheter i atomreaktorer osv.)

- Konsekvensen av et sikkerhetsbrudd i et kontrollsystem kan påvirke fysiske enheter og har potensiale for å sette mennesker, helse, miljø og bedriftens økonomi i fare.

- Man får et "single point of failure", når alt sentraliseres i et offshore kontrollrom. Dette gjør det ekstra viktig å sikre dette punktet.

- Eksisterende kontrollsystem vil ofte benytte flere eldre enheter, med ulike operativsystem og protokoller. Manglende kompatibilitet kan hindre arbeidet med å innføre sikkerhetstiltak i kommunikasjonskanalene.

2.2 Eksempler på angrep

Myke angrep

Uansett hvor sofistikerte metoder som benyttes, vil de mest effektive angrepene likevel være de som er rettet mot det menneskelige elementet. En angriper kommer ofte lenger med utpressing, svindel og bestikkelser enn med rene dataangrep. Det er også ofte menneskelige feil som introduserer ulike former for datatrusler inn i et system, ved å åpne infiserte mailer, koble til infiserte minnepinner, eller rett og slett ved å velge passord som "12345". De viktigste formene for beskyttelse mot denne type angrep er opplæring av personale og begrensninger i systemtilgang.

Mann i midten angrep

Mann i midten angrep er en samlebetegnelse på angrep hvor angriper gir seg ut for å være en part man har tillit til. I motsetning til i dekrypterings angrep, diskutert i seksjon 4.6, der angriperen kun kan lytte etter meldinger mellom de kommuniserende partene, kan en med mann i midten angrep endre meldinger, slette meldinger eller rett og slett finne opp meldinger. Dette gjør disse angrepene mye kraftigere. De to kommuniserende partene vil tro at de kommuniserer med hverandre, men i virkeligheten kommuniserer de begge med en angriper som har mulighet til å endre meldingene til sin egen fordel.

Repetisjons angrep

Repetisjons angrep (En. Replay attacks) er en type mann i midten angrep der en angriper fanger en melding og lagrer den med hensikt om å sende den om igjen senere. Det er to varianter av dette angrepet: En der angriperen kopierer meldingen, men lar den komme frem til mottager uforandret, og en der angriperen fanger opp meldingen slik at den ikke når mottager. Senere kan angriperen sende hele eller deler av meldingen om igjen og lure mottageren til å tro at meldingen kommer fra den opprinnelige avsenderen. For å gjennomføre et slikt angrep trenger ikke angriper å dekryptere meldingen, men må ha en viss oversikt over hva meldingen inneholder. Mange protokoller inkluderer i dag et tidsstempel i alle meldinger for å verifisere at en mottatt melding nettopp ble sendt. Slik unngår de slike angrep.

2.3 Ormer og virus

Ormer er ondsinnet programvare som kopierer seg selv inne på en PC, eller inn i en e-post,og kan slik spre seg fort. En orm vil kompromittere sikkerheten på en pc. Virus er programvare som kopierer seg selv ved å infisere andre programmer, sektorer eller dokumenter på en pc. De er som oftest ondsinnet og kan gjøre stor skade på sensitive data.

Gissel angrep

I senere tid har man sett en ny trend i angrep mot bedrifter, såkalte gissel angrep. Dette er ormer eller virus som krypterer alle filene på en infisert pc. En person som prøver å åpne de krypterte filene vil kun få en feilmelding som sier at filene er uleselige. Angriperen krever så store pengesummer for å låse opp filene igjen. Ofte trues det med å slette filene helt hvis kravene ikke blir innfridd innen et visst tidsrom. Om en bedrift ikke har gode nok backup-løsninger vil dette kunne gi store økonomiske tap, da viktige dokumenter som produkttegninger og testresultater kan gå tapt.

Dos angrep

DoS angrep (Denial of Service angrep), er angrep der en angriper forøker å gjøre en tjeneste utilgjengelig ved å overbelaste systemet som leverer tjenesten. Dette skjer gjerne ved at det sendes mengder av tilkoblings-forespørsler eller forespørsler etter informasjon på en gang.

En datamaskin alene er sjelden nok til å lykkes med et DoS angrep. Derfor benyttes det såkalte "botnet", et nettverk av datamaskiner som alle sender forespørsler samtidig. Maskiner blir rekruttert inn i botnettet ved bruk av ormer og virus, og eierne av maskinene vil ofte ikke være klar over at maskinen er infisert.

Trojanske hester

Såkalte "Trojanere" er ormer eller virus som ikke direkte påvirker den infiserte maskinen, men som sender informasjon om systemet til en tredjepart. Ettersom de ikke gjør skade på den infiserte maskinen, vil eierne ofte ikke være klar over at maskinen er infisert. Samtidig kan sensitive filer som for eksempel passord bli sendt direkte til angriperen.

Kapittel 3

Tiltak

3.1 Brukerautentisering

All sikring av kommunikasjon mellom parter vil være forgjeves, om ikke de kommuniserende partene i seg selv er sikre. Det er derfor et poeng å begrense tilgang til endesystemene, enten det er en server i nettverket, eller en maskin i et kontrollrom. Den mest effektive og mest brukte metoden for dette er ulike former for brukerautentisering. Det er et effektivt middel for adgangskontroll, og kan gjøre det lettere og gi ansatte i en bedrift ulike nivåer av tilgang. Det kan for eksempel være at alle ansatte har lesetilgang på enkelte variabler, slik at det er mulig å følge med på hva som skjer ute i felt, mens det kun er noen utvalgte operatører på bestemte maskiner i kontrollrommet som har skrivetilgang og kan endre fysiske verdier og settpunkter.

Det finnes flere typer brukerautentisering, der den vanligste er et brukernavn i kombinasjon med et selvvalgt passord. Det svake punktet i denne typen autentisering er at brukernavn og passord ofte lagres sammen med andre brukernavn og passord. Om en utenforstående skulle få tak i en slik liste, er plutselig store deler av nettverket åpent. For å unngå dette, bruker mange nå også fysisk identifikasjon i tillegg, i form av f. eks. et personlig adgangskort til et kontrollrom, eller en bank-id til innlogging i nettbanken.

En gylden regel når det gjelder slike restriksjoner er at tilgang til ulike funksjoner bør gis på en såkalt “need to know”-basis, altså at ingen får tilgang til mer enn det er behov for. Dette er ikke kun for å hindre ondsinnede angrep innenfra, men også for å hindre utilsiktede angrep. En e-post som inneholder et virus kan gjøre stor skade om den åpnes av en uvitende bruker med full systemtilgang.

3.2 Profisafe

PROFIBUS DP er en mye brukt standard i kommunikasjon mellom kontrollsystemer og styringsenheter i felt. Den dekker kun de to nederste lagene i ISO/OSI modellen for kommunikasjon, og legger få restriksjoner på det fysiske kommunikasjonsmediet. PROFIBUS DP er ikke robust nok for sikkerhetskritiske systemer, så det ble etterhvert utviklet et tillegg til PROFIBUS-standarden, kalt DP Safety Profile, ProfiSafe. ProfiSafe inneholder en rekke tilleggsfunksjoner for å tilfredsstille de høye kravene til sikkerhetskritisk kommunikasjon. ProfiSafe er utfyllende beskrevet i [1], men noen av de viktigste elementene beskrives nærmere her.

Header

Aller først i alle meldinger plasseres det en preambel. Det er en forhåndsbestemt bitsekvens som brukes til å synkronisere mottageren med timingen til den mottatte pakken. Etter dette følger det en SFD (Start of Frame Delimiter). Det er en kortere bitsekvens som markerer starten på meldingen. Headeren inneholder også lengden på meldingen, samt avsenders og mottagers adresse. Helt til sist i meldingen (etter dataene som skal sendes) finner man også en EFD (End of Frame Delimiter), som markerer at meldingen er slutt.

Sekvensnummer

ProfiSafe bruker et sekvensnummer i alle meldinger som sendes mellom kontrollsystemet og en styringsenhet. Svar-meldingene fra styringsenheten skal inneholde det samme nummeret som var inkludert i den originale meldingen fra kontrollsystemet. På denne måten detekteres tap av meldinger, meldinger som ankommer i feil rekkefølge, samt feil i meldingene. Sekvensnummeret representeres ved en byte som teller fra 1 til 255 før den settes tilbake til 1 igjen. Tallet 0 er reservert til oppstart av systemet, slik at oppstartstilfellet kan håndteres separat fra resten av kommunikasjonsprotokollen. Sekvensnummeret er også et effektivt tiltak for å hindre repetisjons angrep, ettersom det vil gi en feil i sekvensnummeret.

CRC

For å detektere bitfeil som oppstår under overføring av meldinger, benyttes det en CRC (Cyclic Redundancy Check) kode, en syklisk redundant sjekk. Denne koden genereres ved å be-

nytte en enkel nøkkel som er felles for både sender og mottager. Dataene som skal sendes divideres bitvis på denne nøkkelen gjentatte ganger til det står igjen en rest som er mindre enn nøkkelen. Denne resten legges til meldingen før sending. Når meldingen mottas vil mottager bruke den felles nøkkelen og gjøre samme utregning. Om resten som genereres ikke stemmer med den som følger med meldingen, er det oppdaget en bitfeil, og mottager ber om en resending av meldingen.

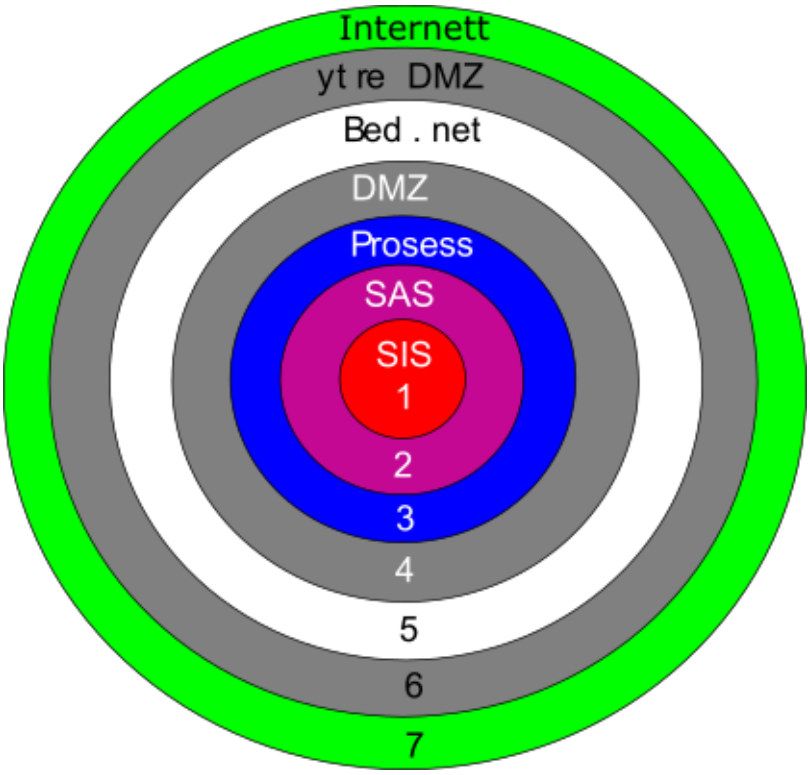
I ProfiSafe benyttes noen av de felles sikkerhetsparameterne som f. eks. kodenavn til enheter og SIL nivå til å generere en CRC1 nøkkelkode hos både sender og mottager. Denne er i de fleste tilfeller 32 bit lang, ettersom CRC-32 er den mest brukte standarden. Benytter man CRC-32, vil det være mulig å detektere alle 1 eller 2 bits feil i de første $2^{32} - 1$ posisjonene [4]. CRC1 nøkkelkoden benyttes så som dividend over hele den sikkerhetskritiske informasjonen og genererer en rest ofte referert til som en CRC2 kode. Denne koden blir så lagt til meldingen før sending. Ettersom sender og mottager har tilgang til de samme sikkerhetsparametrene, vil den genererte CRC1 nøkkelkoden være den samme, og mottager kan enkelt generere sin egen CRC2 for å sammenligne.

En CRC er et effektivt virkemiddel for å oppdage bitfeil som er et resultat av tilfeldig datamodifikasjon, ettersom den kan detektere 99.9985% av alle slike feil [1]. Det er derimot ikke god beskyttelse mot data som er modifisert med hensikt. Det er forholdsvis enkelt og bruke en melding og en medfølgende CRC kode til å finne nøkkelen som ble brukt i utregningen. Om data i meldingen endres med hensikt, er det mulig å finne og bruke denne nøkkelen til å regne ut en ny CRC kode som stemmer overens med innholdet i meldingen og å legge til denne i stedet for den opprinnelige CRC koden. Mottagerens utregning vil da stemme med CRC koden i den mottatte meldingen, og endringene forblir uoppdaget.

Kontroll/Status byte

ProfiSafe inkluderer også en Kontroll/Status byte i alle meldingene. I meldinger fra styringsenheten er dette konfigurert som en status-byte, og gir informasjon om forskjellige feil som er oppdaget, om systemet er gått i sikker tilstand, og om meldingen inneholder oppdaterte parameterverdier.

I meldinger fra kontrollsystemet er dette en kontroll-byte. Her er det kun bit 0 som er relevant, det brukes om styringsenheten trenger nye parameterverdier. I det tilfellet sendes parametre for sikker tilstand.



Figur 3.1: Illustrasjon av "løkmodellen", som presentert i [6]

Frame checking sequence

FCS, Frame checking sequence, er en CRC kode som genereres over hele meldingen, og ikke bare den sikkerhetskritiske informasjonen. Det vil si at den også inkluderer feltene for avsender/mottager adresse, start og stopp -sekvenser, osv.

3.3 Nettverksstruktur

En av de største utfordringene med bruk av internett til kontrollformål er at det åpner for uautorisert tilgang utenfra. Et viktig tiltak er derfor å lage skiller mellom nettverkene internt i bedriften. Slik kan man hindre at uvedkommende som får tilgang til bedriftsnettverket får direkte tilgang til kontrollsystemet. Det er ønskelig å skille ulike deler av systemet for å oppnå "beskyttelse-i-dybden", ofte representert som en "lagmodell" eller "løkmodell" som illustrert i figur 3.1. Bedriftsnettverket kan bestå av mange installasjoner, men prosessnettverket vil typisk bare være en installasjon og bør separeres fra bedriftsnettverket så mye som mulig.

DMZ

Modellen i figur 3.1 bruker to demilitariserte soner (DMZ) for å dele opp nettverket. Det er logiske barrierer mellom ulike deler av nettverket, realisert ved f. eks. brannmurer. Sterk adgangskontroll og brukerautentisering kan også inngå i DMZ. I SIS (Sikkerhets Instrumenterte Systemer) er det også risikabelt å åpne for direkte manipulasjon av prosessvariabler. Da kan DMZ-en inneholde en server som tar imot instruksjoner fra bedriftsnettverket. Enheter i PCS (Process Control System) på andre siden av DMZ-en kan sjekke periodisk etter oppdateringer i prosessvariablene. Adgangskontroll kan også implementeres i en DMZ. Enkelte deler av systemet har ikke behov for endring i prosessvariabler, og bør derfor kun ha lesetilgang. En DMZ kan i et slikt tilfelle ha toveis kommunikasjon med et PCS og enveis kommunikasjon (kun lesing) med et SIS. En slik løsning kalles en diodeløsning [6].

Brannmurer

For å realisere en DMZ er man avhengig av brannmurer. De blir benyttet både internt i nettverk for å realisere DMZ-er og i periferien av et nettverk som en barriere mot omverdenen. Brannmurer er programmer eller maskinvare (eller en kombinasjon) som sorterer nettverkstrafikken utifra forhåndsbestemte parametre, og filtrerer ut trafikk som ikke har de rette egenskapene. Det finnes flere typer brannmurer definert etter hva de filtrerer på. Pakkefiltrerende brannmurer filtrerer på innholdet i datapakkene som blir mottatt, for eksempel avsenders adresse. Applikasjonsfiltrerende brannmurer vil se på datafeltet i datapakkene, og hva som slipper gjennom er applikasjonsbestemt. Disse brannmurene er tregere enn de pakkefiltrerende brannmurene ettersom de bruker lengre tid på å prosessere hver pakke. De to typene brannmurer brukes derfor ofte i serie, hvor en pakkefiltrerende brannmur gjør en grov sortering først, og en applikasjonsfiltrerende brannmur sjekker de pakkene som kom gjennom den første filtreringen mer nøye.

Hardening

Hardening består i å fjerne eller deaktivere fysiske elementer som ikke er nødvendig for formålet. Et eksempel kan være å deaktivere alle internett-porter på en enhet utenom de man vet det er nødvendig å bruke, eller å deaktivere det trådløse nettverkskortet om man vet man skal koble til internett ved bruk av en fysisk kabel. Deaktivering av programmer som kom-

0	0	0	0	0	0	0	“d”	“a”	“t”	“a”
A	A	A	A	A	A	A	B	B	B	B

Tabell 3.1: Byte allokert til henholdsvis variabel A og variabel B.

“s”	“i”	“k”	“k”	“e”	“r”	“h”	“e”	“t”	“t”	“a”
A	A	A	A	A	A	A	B	B	B	B

Tabell 3.2: Bufferoverflyt fra variabel A.

muniserer med internett, f. eks. e-post klienter og nettlesere er også fornuftig.

3.4 Bufferoverflyt

En av de vanligste feilene som gjøres i henhold til datasikkerhet er manglende inputvalidering. Dette kan føre til bufferoverflyt eller i verste fall kommandoinjeksjon. Ved bufferoverflyt blir det forsøkt å lagre en variabel som er større enn det minnet som er allokert til denne variabelen. Variabelen vil da overskrive noe minne på utsiden av den allokerte bufferen. Om denne delen av minnet var benyttet til noe annet, vil det da inneholde feil verdier. I tabell 3.1 ser vi at variabel A har 7 byte allokert, som ligger ved siden av 4 byte som er allokert til variabel B. Ordet “sikkerhet” fyller 9 byte, mer enn det som er allokert til variabel A, og flyter over i minnet til variabel B, illustrert i tabell 3.2. Disse to tabellene er basert på to tabeller fra [5].

Om det ikke er god nok inputvalidering, er systemet også utsatt for kommandoinjeksjon. Om det eksisterer en kjent bufferoverflyt kan en angriper utnytte denne svakheten til å laste kjørbar kode inn i minnet. Ved å avslutte injeksjonen med f. eks. et semikolon eller en parentes kan angriperen i noen tilfeller klare å kjøre koden. På denne måten er det mulig å introdusere virus eller annen skadelig programvare til et system uten at det blir stoppet i en brannmur.

Manglende inputvalidering kan også åpne for det motsatte av kommandoinjeksjon. En angriper kan etterspørre en variabel, og spesifisere at den er lengre enn det den egentlig er. Om en angriper etterspurte variabel A i tabell 3.1 og så spesifiserte at det skulle hentes 11 byte, ville angriperen så få tilgang til hele variabel B. Dette kan brukes for å få tilgang til sikkerhetskritiske filer, som f.eks databaser hvor det lagres krypteringsnøkler, og er viktig å beskytte seg mot.

Kapittel 4

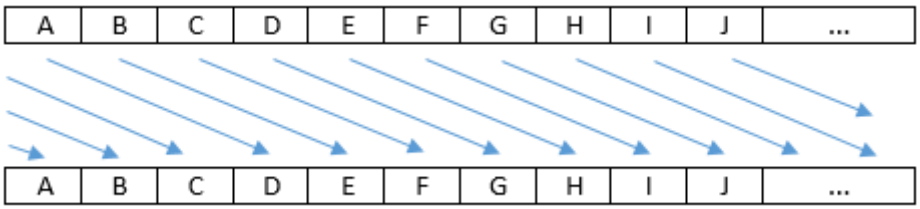
Kryptering

Innholdet i dette kapittelet er i hovedsak hentet fra [4].

4.1 Introduksjon

Kryptering er en viktig del av sikkerhetstiltakene som benyttes for sikker kommunikasjon over det åpne internett. Formålet er å gjøre meldinger uleselige for utenforstående, samt å hindre utenforstående i å endre meldinger til sin egen fordel. En av de første formene for kryptering som ble benyttet er det såkalte "Cæsar sifferet" (En. "Caesar cipher"), oppkalt etter den første dokumenterte brukeren: Julius Cæsar. Det krypterer tekst ved å bytte ut alle bokstavene i teksten med bokstaven et visst antall plasser frem, som oftest tre. På denne måten vil ordet "datasikkerhet" krypteres til den ubetydelige teksten "gdwdvlnnhukhw". Dette er illustrert i figur 4.1. Å dekryptere et Cæsar siffer er trivielt med moderne teknologi og dagens regnekraft, men denne utviklingen åpner samtidig for stadig mer avanserte krypteringsalgoritmer.

Krypteringsalgoritmer må kunne garantere at kun sender og mottager har den nødvendige informasjonen til å dekryptere meldinger når de kommer frem. I tilfellet med "Cæsar siffe-



Figur 4.1: Illustrasjon av et "Cæsar siffer"

ret" er denne informasjonen hvor mange plasser i alfabetet hver bokstav er flyttet. I moderne krypteringsalgoritmer er denne informasjonen gjerne en bitsekvens som kun de involverte partene har tilgang til, og den refereres til som en nøkkel. Nøklene byr på store utfordringer, da en krypteringsalgoritme aldri kan bli sikrere enn distribueringen av nøkler mellom de involverte partene. Dette er ofte kalt krypteringens paradoks, ettersom man for å dele hemmelig informasjon først må dele annen hemmelig informasjon.

4.2 Krypteringsalgoritmer

En av de største utfordringene innen kryptering er at en algoritme som har blitt kompromittert en gang, i fremtiden vil være nærmest ubrukelig siden samme metode kan benyttes igjen. Resultatet av dette, er at kryptografi er i stadig utvikling og at protokoller som benytter kryptering må være lett vedlikeholdbare og fleksible. Om protokoller og systemer har en høy grad av modularitet, er det enklere å skifte ut utdaterte komponenter.

For å kryptere en tekst, benyttes som regel en av to hovedtyper algoritmer: strømme-kryptering (En. stream cipher) og blokk-kryptering (En. block cipher). Strømme-kryptering benyttes gjerne der det sendes en konstant strøm av byte over et nettverk, og er laget slik at man kan kryptere og dekryptere deler av meldingene underveis. Blokk-kryptering deler teksten som skal sendes inn i blokker før kryptering og sender blokkene separat. Dette er mer vanlig i meldingsbasert kommunikasjon.

4.3 Krypteringsblokker

I blokk-kryptering benyttes en algoritme til å generere en krypteringsblokk utifra en krypteringsnøkkel og en blokk tekst som skal krypteres. Algoritmen som benyttes må være invertibel, for å muliggjøre dekryptering. Den vanligste formen for blokk-kryptering er itererte blokk-siffer (En. iterated block ciphers), der en tekstblokk av en bestemt lengde blir omgjort til en kryptert tekstblokk av samme lengde. Dette skjer gjennom en serie av reverserbare operasjoner kalt runder (En. rounds), som varierer mellom de ulike algoritmene og er avhengig av krypteringsnøkkelen. Runder kan være sammensatt av blant annet permutasjoner, addisjoner, substitusjoner og gjerne XOR operasjoner med en egen rundenøkkel (En. roundkey) utledet fra krypteringsnøkkelen. Et SPN (Substitution-Permutation Network) er et viktig

iterert blokk-siffer som kun benytter substitusjoner og permutasjoner. Ulike blokk siffer benyttes sammen med forskjellige operasjonsmodus (En. Mode of operation). Disse har stor innvirkning på sikkerheten til et bestemt blokk siffer, da et sikkert blokk siffer ikke vil være sikkert om det brukes sammen med et usikkert operasjonsmodus og omvendt. Ulike operasjonsmodus er nærmere omtalt i seksjon 7.2.

4.4 Tilfeldighet

Om den samme teksten krypteres mer enn en gang, er det ønskelig at resultatet ikke blir det samme hver gang. Om lik tekst alltid gir lik kryptert tekst vil teksten være uleselig, men mønstre i teksten vil være tydelige. Som et eksempel vil ordet "bananer" kryptert med Cæsar sifferet fra tidligere bli "edqdqhu". Det er tydelig at d'ene og q'ene er to like bokstaver. Om en da tar hensyn til at det bør være en viss mengde vokaler i teksten, og at e, a og r er tre av de mest brukte bokstavene i det norske språk, er det forholdsvis enkelt å komme frem til det riktige ordet. Dette illustrerer også et annet prinsipp som benyttes mye i kryptoanalyse; frekvensanalyse. Det er mulig å sammenligne frekvensen av hver bokstav i den krypterte teksten med frekvensen av hver bokstav i vanlig tekst, for så å forsøke å bytte ut bokstaver med omtrent samme frekvens. Moderne krypteringsalgoritmer er generelt for avanserte til at dette er mulig, men det er likevel mye brukt innen kryptoanalyse. En stor utfordring innen kryptoanalyse er å få datamaskiner til å gjenkjenne en riktig nøkkel, og frekvensanalyse kan benyttes til å vurdere ulike løsninger som er funnet.

For å unngå synlige mønstre i den krypterte teksten, er det viktig å inkludere et tilfeldig element. Det er også viktig å kunne garantere at dette elementet er *helt*-tilfeldig, og ikke kun *semi*-tilfeldig, da semi-tilfeldighet vil gjøre krypteringsalgoritmen mindre sikker. Datagenerering av helt tilfeldige tall, uten noen form for bias, er utfordrende. Noen av de bedre løsningene er algoritmer som produserer semi-tilfeldige tall, men som blir initialisert med en så tilfeldig variabel som mulig, for eksempel skalert støy fra en ubrukt port.

Den tilfeldige initialiserings-variabelen (IV) vil kun bli benyttet en gang. Den må derfor sendes i klartekst sammen med den krypterte teksten, slik at det er mulig for mottageren å benytte den tilfeldige variabelen til å dekryptere den mottatte meldingen.

4.5 Autentisering

Kryptering med blokk siffer kan garantere konfidensialitet, men kan ikke alene garantere integritet. Med andre ord, en kryptert melding vil være uleselig for utenforstående, men det er mulig at meldingen har blitt endret underveis uten at det oppdages av mottager. Dette åpner blant annet for valgte siffertekst angrep som beskrevet i seksjon 4.6. For å sikre integriteten i meldingene som mottas, er det viktig å bruke autentisert kryptering. Til dette formålet benyttes det en MAC (Message Autentication Code). En MAC-kode er basert på en melding som skal sendes, og genereres ved bruk av kryptografiske primitiver som "hash"-funksjoner eller blokk-siffer. Mottageren av meldingen benytter så MAC-koden til å verifisere at innholdet i meldingen ikke har blitt endret.

I likhet med den tilfeldige initialiserings-variabelen må også MAC-koden sendes sammen med meldingen den er generert over. Det finnes tre måter å gjøre dette:

1. **Kryptering og MAC:** Krypter klarteksten, generer en MAC av klarteksten og legg MAC-koden til etter den krypterte teksten før den sendes.

2. **Kryptering så MAC:** Krypter klarteksten, generer en MAC av den krypterte teksten, og legg MAC-koden til etter den krypterte teksten før den sendes.

3. **MAC så kryptering:** Generer en MAC av klarteksten, legg den til klarteksten og krypter begge deler sammen.

Testresultater publisert i [2] viser at alternativ 2, kryptering så MAC, er det eneste sikre alternativet. I de to andre alternativene genereres MAC-koden på basis av klarteksten. Om en angriper har informasjon om hvordan MAC-koden blir generert, kan det gi informasjon om klarteksten. Ved bruk av alternativ 3 er det i verste fall mulig at en angriper får informasjon om krypteringsnøkkelen benyttet for å kryptere MAC-koden.

Nødvendigheten av autentisert kryptering ble ikke formelt bevist før i år 2000 [2]. Mangel på oppdateringer gjør at det derfor er mange eldre systemer som ikke inkluderer autentisering i krypteringsalgoritmene. Implementasjon av autentisering som tillegg til vanlig kryptering er utfordrende og åpner for mange feil. Derfor har det i det siste dukket opp flere nye operasjonsmodus for blokk-kryptering som kombinerer kryptering med en autentiseringsfunksjon. Ulike operasjonsmodus er diskutert nærmere i seksjon 7.2.

4.6 Dekrypterings angrep

Det finnes flere typer dekrypterings angrep. De forskjellige variantene skilles ofte utifra hvilke ressurser en angriper har tilgang til. Den enkleste formen for angrep er et såkalt "brute force" angrep der angriper kun har tilgang til kryptert tekst, og prøver å finne originalteksten eller krypteringsnøkkelen ved å prøve alle mulige krypteringsnøkler til noe gir fornuftige resultater. Dette er kalt den enkleste formen for dekryptering, ettersom den krever veldig lite informasjon, og ettersom det som regel er uproblematisk og få tak i krypterte meldinger ved å lytte på andres kommunikasjon. Det er på den andre siden en lite effektiv metode, ettersom den er tidkrevende og krever store mengder regnekraft for å gi resultater innen et fornuftig tidsvindu. Ettersom et brute force angrep kun trenger å få tilgang til krypterte meldinger, kan man ikke lage krypteringer som gjør dekryptering med denne metoden umulig. Det man sikter mot i beskyttelse mot slike angrep, er at dekryptering skal ta så lang tid at informasjonen som hentes ut ikke lenger vil være relevant.

Et krav for å lykkes med en brute force tilnærming, er å gjenkjenne den riktige krypteringsnøkkelen når den er funnet. Om det er lite kryptert tekst tilgjengelig, kan det være at det finnes flere ulike krypteringsnøkler som gir meningsfulle meldinger. Da vil angriperen ha behov for mer informasjon. I enkelte tilfeller, f. eks. når det benyttes offentlige krypteringsnøkler (En. public keys) som er gjeldende i en lengre tidsperiode, er det realistisk at en angriper vil få tilgang til en kryptert melding og korresponderende originaltekst. Da kan angriperen med sikkerhet gjenkjenne den riktige nøkkelen når den testes, men tidsbruk og tilgjengelig regnekraft er fremdeles et problem.

Alle dekrypteringer som er raskere enn en ren brute force løsning kategoriseres som et kryptografisk "break". For å få til et break må en gjerne ha tilgang til mer informasjon enn kun kryptert tekst. Ulike angrep kan ofte kategoriseres utifra hvilken informasjon angriperen har tilgjengelig. Det foregående eksempelet er kjent som et "kjent klartekst angrep" (En. known plaintext attack). En ren brute force løsning er til sammenligning et "kjent siffertekst angrep" (En. known ciphertekst attack).

Det mest effektive for å redusere tidsbruken, er å snevre inn mengden aktuelle nøkler. For å hente informasjon om nøkkelen som brukes, kan en angriper bruke en valgt tekst og prøve å få offeret til å kryptere denne, f. eks. ved å plante teksten i en vanlig melding. Angriperen velger da en tekst som forventes å gi informasjon. Dette kategoriseres som et "valgt klartekst angrep" (En. chosen plaintext attack). Det motsatte angrepet, hvor en angriper forsøker å få

offeret til å dekryptere en spesielt valgt melding er naturlig nok et “valgt siffertekst angrep” (En. chosen ciphertext attack).

En alternativ fremgangsmåte for å redusere antall nøkler å teste er å forsøke å dekryptere kryptert tekst med tilfeldige nøkler for så og analysere resultatet. Dette kan gi informasjon om den virkelige nøkkelen.

Del III

Oppgave

Kapittel 5

Arduino

Platformen Arduino brukes i dette prosjektet av flere grunner, hvorav den viktigste er å skape et realistisk scenario. Den inneholder en mikrokontroller som ikke bruker et av de mest brukte operativsystemene som f. eks. Windows, men et mer lavnivå språk. Dette vil ofte være tilfelle i instrumenteringssystemer.

En annen grunn til å benytte Arduino er at det er "open source", noe som vil si at alt kodemateriale er tilgjengelig for bruk. Det gjør at de forskjellige kretskortene og komponentene er godt dokumentert og forholdsvis enkle og bruke.

5.1 Komponenter

Dette er en oversikt over fysiske komponenter som ble benyttet i prosjektet:

- Arduino Mega 2560 brett med en ATmega2560 kjerne
- Arduino Ethernet Shield V2
- Power over Ethernetmodul
- 2 knapper
- 2 LED-lys (forskjellige farger)
- 2 motstander 220 Ω
- 2 motstander 10 K Ω
- Koblingsbrett

- Kategori 5 ethernet kabel
- USB kabel
- Internett-switch
- "Jumper wires" (små ledninger) i assorterte farger

Det ble brukt en Arduino "Mega 2560", ettersom den hadde endel større minne enn den mer brukte og noe billigere Arduino "Uno". (Se appendiks B for detaljerte spesifikasjoner.) I tillegg ble det benyttet et Arduino "Ethernet Shield". Det er et spesialisert kort som inneholder en standard RJ45 ethernet port og forenkler tilkoblingen mellom en Arduino og ethernet. Dette kortet benytter også en "Power over ethernet"-modul, en modul som trekker strøm fra ethernet-tilkoblingen. Denne er nødvendig da transformatoren på Arduino Mega 2560 ikke klarer å levere nok spenning til å drive både Arduino Mega 2560 og Arduino Ethernet Shield uten å bli overbelastet ved bruk over lengre tid.

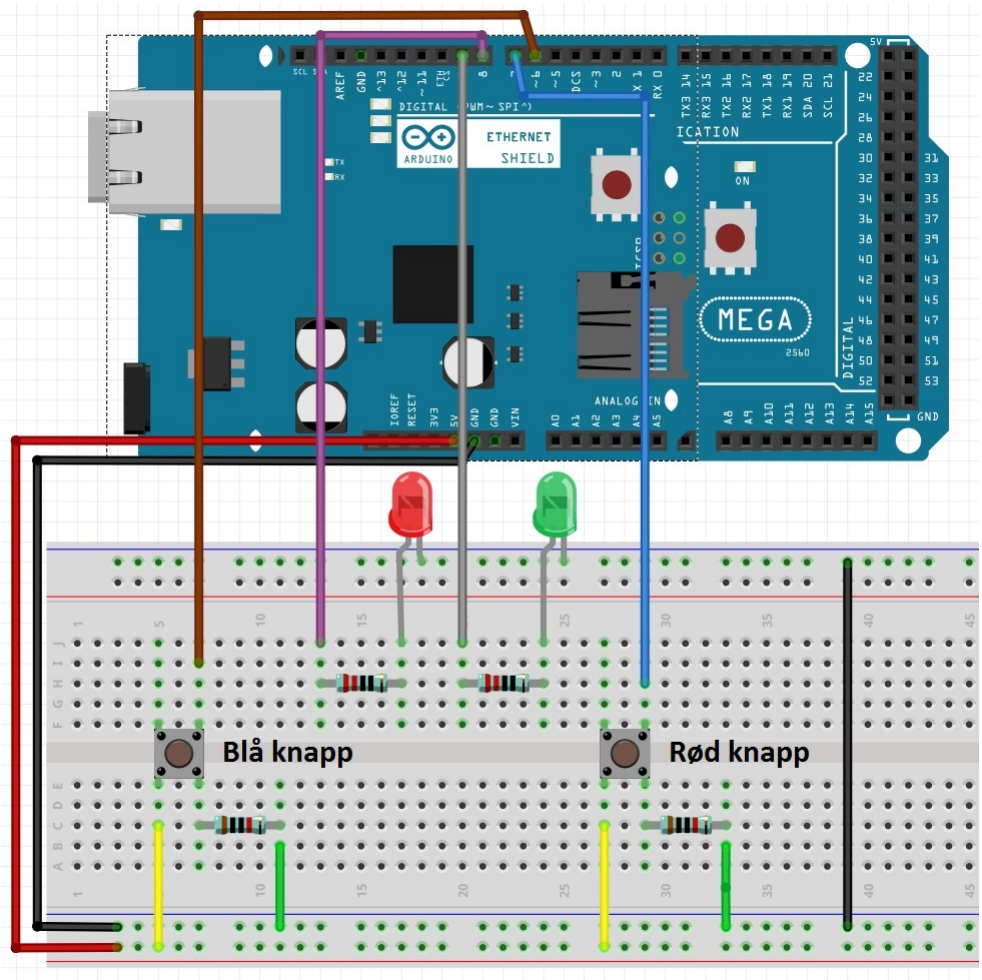
5.2 Oppsett/koblinger

Figur 5.1 viser en oversikt over koblingene som er gjort og komponentene som er brukt. Kretsen er kun laget for å virke som en indikator på hvor godt Arduino-en klarer å opprettholde kommunikasjon og funksjonalitet under forskjellige typer eksterne angrep. Funksjonaliteten er derfor holdt så simpel som mulig.

I kretsen er det en rød og en grønn LED. Det er kun en LED som lyser av gangen, og en blå knapp brukes til å skifte mellom rødt og grønt lys. En rød knapp brukes for å indikere at systemet skal gå i sikker tilstand. Da skrus begge LED lysene av. Disse funksjonene skal også kunne utføres ved hjelp av kommando-er fra PCS.

All kode som håndterer knappene og LED-lysene på koblingsbrettet er implementert i et eget bibliotek i programmerings-språket C++. Dette er for å gjøre det enklere å få oversikt over koden som omhandler selve kommunikasjonen med PCS.

Arduino-en er koblet til internett via en ethernet-kabel, og kommuniserer med PCS over internett. Den er i tillegg koblet til den samme PC-en med en USB-kabel. Denne tilkoblingen er kun ment som et diagnostiseringsverktøy, da den tillater å skrive variabler fra Arduino-en til en skjerm, og har ingen innvirkning på kommunikasjonen med PCS. USB-kabelen leverer



Figur 5.1: Fysisk oppsett på Arduino og koblingsbrett

også strøm til Arduino-en, men denne strømtilførselen kan enkelt erstattes med et batteri som gir mellom 7V og 12V.

5.3 Inputvalidering

For å beskytte mot kommandoinjeksjon og bufferoverflyt som beskrevet i seksjon 3.4, er det viktig å benytte inputvalidering. Det er spesielt viktig å verifisere at størrelsen på mottatte variabler ikke overskrider størrelsen av den allokerte bufferen. Ikke statiske variabler i programkoden på Arduinoen er nesten utelukkende lagret som booleans eller byte. Byte formatet i Arduino-kode er konstruert slik at en variabel vil lagres modulo 256. For eksempel vil tallet 257 lagres som 1. En variabel som har en for stor verdi vil derfor ikke føre til bufferoverflyt. En boolean er enten 0 eller 1, og et forsøk på å lagre noe annet i en slik variabel vil ikke fungere.

I de tilfeller der variablene er større enn en byte, lagres de som en liste av byte. Ettersom hver byte er sikret mot bufferoverflyt, er det kun nødvendig å verifisere at antall byte som lagres er korrekt. Meldingene som mottas fra PCS har en fast lengde, så dette sikres ved å kun lese det riktige antallet byte inn i meldingsbufferen. Om en angriper har utvidet et felt i midten av meldingen, vil ikke de riktige elementene være med, det blir registrert en feil, og meldingen blir ignorert. Om det er lagt til noe ekstra på slutten av en melding, vil meldingen bli for lang, og det som er lagt til vil ikke bli lest inn i meldingsbufferen.

En mottatt melding vil heller aldri bli benyttet direkte i en funksjon. Hver gang noe av innholdet i en mottatt melding benyttes i en funksjon, blir et bestemt antall byte ekstrahert fra meldingen og lagret i en tilhørende buffer. Den konstante variabelen som indikerer hvor mange byte som skal ekstraheres, er den samme som benyttes til initialisering av bufferen. Slik sikres det at bufferen har stor nok kapasitet. Den samme fremgangsmåten er også benyttet i PCS.

PCS kan også ta imot kommando-er fra bruker. Her finnes det kun fem gyldige kommando-er:

- “green on” -> Skru på grønt lys
- “red on” -> Skru på rødt lys
- “failsafe -> Aktiver sikker tilstand (Systemet stenger ned)

- “failsafe off” -> Deaktiver sikker tilstand (Systemet starter opp igjen)
- “current state” -> Print nåværende systemtilstand til konsoll

Om det blir gjort forsøk på å gi annen input vil det kun bli printet en feilmelding som opplyser om hvilke kommando-er som er gyldige.

Kapittel 6

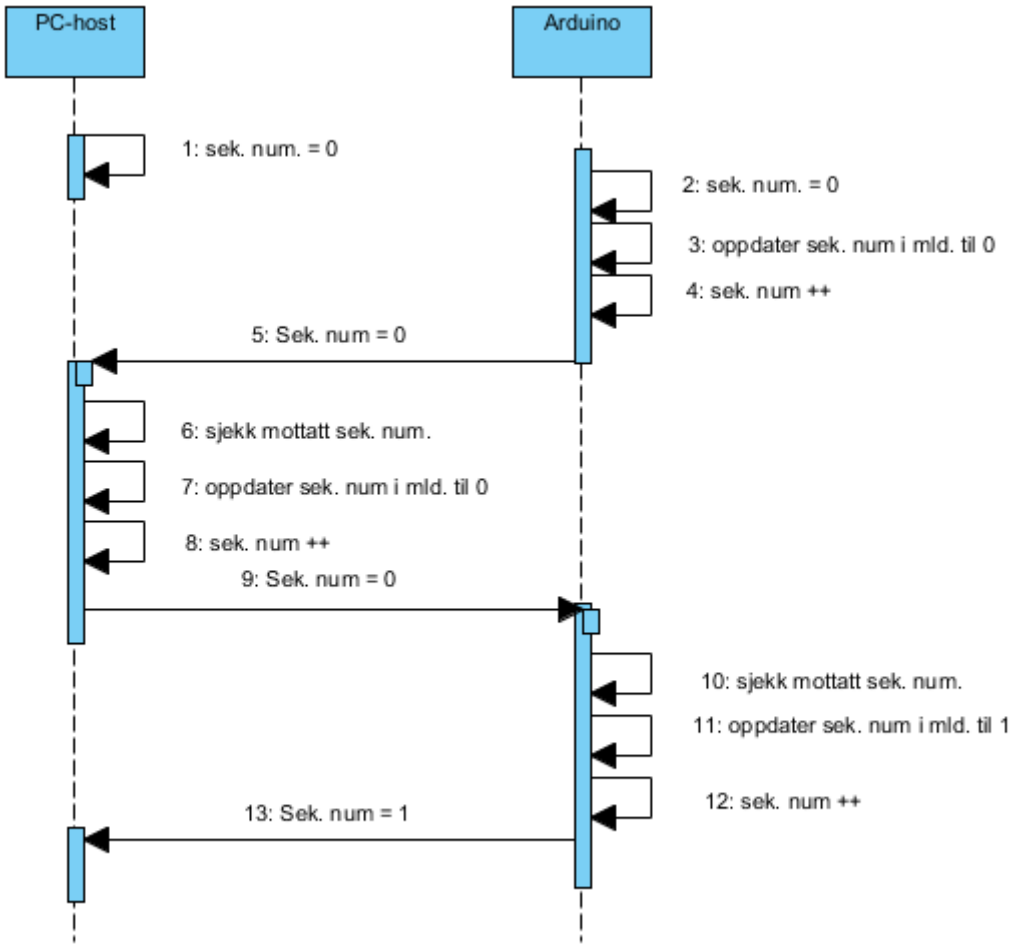
Kommunikasjon

Programmeringsspråket Python ble benyttet til å programmere det andre endepunktet i kommunikasjonskanalen, PCS. Det ble valgt ettersom det har gode innebygde funksjoner for å kommunisere over internett, og fordi det finnes gode biblioteker for kryptografi tilgjengelig. For å være mest mulig oppdatert ble det benyttet den seneste versjonen av Python som eksisterer per i dag, versjon 3.5.1. Dette gjorde prosjektet mer utfordrende, ettersom store deler av dokumentasjonen og bibliotekene for Python fremdeles er skrevet for versjon 2.7 eller tidligere.

6.1 Profisafe

Kommunikasjonen mellom PCS og styringsenhet tar utgangspunkt i kommunikasjonsprotokollen ProfiSafe som beskrevet i kapittel 3.2. Ettersom ProfiSafe er konstruert for å fungere alene på isolerte systemer, er det gjort endel endringer i implementasjonen av de ulike delene, men funksjonaliteten er mye den samme. Det er også gjort enkelte endringer for at implementasjoner gjort i styringsenheten skal være kompatible med funksjonalitet i PCS.

CRC generering inkluderer her ikke bruk av nøkler eller andre variabler, kun de data det skal genereres CRC over. Formen for CRC som brukes er en mye brukt implementasjon, og den CRC-en som genereres er konsistent med den som bl.a brukes for ZIP-filer. Dette gjorde det mye enklere å teste funksjonaliteten, da det finnes ulike verktøy for kalkulere slike CRC-koder. Det er også denne formen for CRC som genereres av den innebygde CRC-funksjonen i Python. Arduino har ingen offisielle biblioteker for å generere CRC-kode, men det finnes flere implementasjoner tilgjengelig. Biblioteket som ble benyttet i denne oppgaven kan finnes



Figur 6.1: Sekvensdiagram over oppstart av arduino og host

her: https://github.com/bakercp/CRC32.

Sekvensnummeret er benyttet på samme måte som i ProfiSafe protokollen, med det unntaket at verdien null ikke kun er reservert til oppstart av systemet, men benyttes hver gang kommunikasjonen blir gjenopprettet etter et brudd. Dette ble gjort for å kunne benytte samme prosedyre for tilkobling både under oppstart av systemet, og når det ble forsøkt å gjenopprette kommunikasjon etter et brudd. Hvordan sekvensnummeret sendes og oppdateres er illustrert i figur 6.1.

ProfiSafe inkluderer en byte for status/kontroll, mens det her er valgt å benytte flere byte for forskjellige tilstander.

Antall byte	7	1	6	6
Innhold	Preambel	SFD	Destinasjon MAC	Mottager MAC
Antall Byte	4	2	Variabel lengde	4
Innhold	Tag(valgfritt)	Ethernet type	Data	FCS

Tabell 6.1: Fullstendig oversikt over en melding i ethernet protokollen [11]

.

6.2 Ethernet protokoll

ProfiSafe protokollen er laget for å fungere alene, men er i denne oppgaven brukt i sammenheng med ethernet protokollen. Denne overlapper med ProfiSafe på flere områder, noe som gjør at det ikke er nødvendig å implementere hele ProfiSafe protokollen. Ethernet inkluderer bl. a. sender og mottagers MAC-adresse (Medium Access Control-adresse), FCS, en preambel, og en SFD, en sekvens som indikerer starten av den faktiske meldingen. Det som gjenstår er: prosessvariabler, CRC, sekvensnummer og en status/kontroll byte. Oppsettet i ethernet er illustrert i tabell 6.1. Elementene som gjenstår plasseres i feltet markert "Data".

6.3 Meldingsoppsett

Alt som er beskrevet i denne seksjonen plasseres i feltet merket "Data" i oversikten over ethernet protokollen 6.1. Meldingsoppsettet er illustrert i tabell 6.2.

Den første byte-en i datafeltet er lengden på feltet. Intensjonen er at mottager skal kunne ta inn en byte, lese av lengden, og så ta inn det oppgitte antall byte. Dette er foreløpig ikke nødvendig ettersom meldingslengden er konstant, men om det på et senere tidspunkt skal implementeres logikk for å f. eks. distribuere krypteringsnøkler vil lengden på meldingene variere.

I begynnelsen var det tenkt at funksjonskoden i den neste byten skulle indikere at PCS eller styringsenhet hadde oppdaget en bitfeil i en mottatt melding og at den forrige meldingen skulle resendes. Dette ble senere vurdert som lite hensiktsmessig, da det innen meldingen ble sendt på nytt ville finnes nye oppdaterte prosessvariabler å sende. Istedet ble det implementert logikk for å ignorere innholdet i en melding hvor det var oppdaget feil, men likevel registrere at en feil hadde oppstått. Dette feltet har derfor ingen betydning i koden som den er, men er beholdt slik at det i en eventuell utvidelse av koden vil kunne brukes til å skille ulike typer meldinger fra hverandre.

Prosessvariabelen er det feltet som styrer lysene i styringsenheten. I meldingene som sendes fra PCS til styringsenheten er dette et tall som indikerer hvilken tilstand lysene skal settes i. Tallet kan også indikere at styringsenheten skal gå inn i eller ut av sikker tilstand. Prosessvariabelen fra PCS bestemmes hovedsaklig av bruker-input, men vil også indikere at systemet skal gå i sikker tilstand om det blir oppdaget feil.

I meldinger fra styringsenheten til PCS vil denne byten inneholde oppdatert status fra styringsenheten. Ettersom det er mulig å endre tilstanden til styringsenheten manuelt er det nødvendig å sende oppdateringer tilbake til PCS. I et kontrollsystem vil det være ønskelig å følge med på ulike sensorverdier o.l., så det er nødvendig å sende oppdatert informasjon tilbake. Tallene 1-4 representerer de ulike tilstandene som kan sendes:

1. Grønt lys på
2. Rødt lys på
3. Sikker tilstand aktivert (Systemet stenger ned)
4. Sikker tilstand deaktivert (Systemet starter opp igjen)

Når en melding mottas i PCS blir den nye statusen lagret slik at siste oppdatering kan presenteres for brukere ved behov. Den lagrede statusen blir også benyttet til å stoppe ulovlige operasjoner. Om en bruker f. eks. forsøker å skru på det røde lyset mens sikker tilstand er aktivert vil det kun dukke opp en melding som minner brukeren på at systemet er i sikker tilstand, og at dette derfor er en ulovlig operasjon.

Oppdateringsfeltet indikerer om prosessvariabelen som mottas er en oppdatert ny verdi eller ikke. Om dette feltet er null, vil innholdet i prosessvariabelen ignoreres.

Om det oppdages en feil i en mottatt melding ved, f. eks. at CRC-koden eller sekvensnummeret er feil, vil en feilteller hos mottageren inkrementeres. Så vil feltet “Error” i meldingen som sendes tilbake, settes til 1 for å indikere at det ble oppdaget en feil. Om en slik melding mottas, vil dette også føre til inkrementering av feiltelleren hos mottageren.

Flere typer blokk-kryptering (se seksjon 4.3) krever at alle blokkene som skal krypteres er av samme lengde. Dette oppnås ved “padding”, altså ved å utvide den siste ufullstendige blokken med nuller (i enkelte tilfeller tilfeldige tall). I meldingsbasert kommunikasjon betyr det at alle meldinger som er mindre enn blokk-størrelsen for kryptering må utvides med padding. I dette tilfellet er den delen av meldingen som skal krypteres utvidet til 128 bit (16

Byte	0	1	2	3	4
Innhold	Lengde av mld.	Funksjonskode	Prosessvariabel	Oppdatering	Error
Byte	5	6-9	10-15	16-31	32-48
Innhold	Sekvensnummer	CRC	Padding	IV	MAC-kode

Tabell 6.2: Fullstendig meldingsoppsettet som illustrerer hvilke byte som benyttes til hva.

byte). Om lengden av meldingen skal sendes ukryptert, må padding-en utvides med en byte for at meldingen skal bli riktig størrelse.

"IV" og "MAC-kode" er elementer som benyttes i kryptering men som sendes i klartekst. De er derfor lagt til etter padding. De er nærmere omtalt i henholdsvis seksjon 4.4 og 4.5.

6.4 Robusthet

Det er svært viktig at kommunikasjonen mellom PCS og SIS er robust. Ikke bare må det gjøres tiltak for å unngå feil, men eventuelle feil som skulle oppstå bør håndteres på en slik måte at de ikke påvirker systemsikkerheten. Systemet bør feile på en sikker måte (En. failsafe). Dette kan bety ulike ting avhengig av hva systemet gjør. På en oljeplattform vil det i de fleste tilfeller være å stenge ned en prosess der det er oppdaget feil, mens i et passasjerfly eller helikopter er det selvsagt ikke et alternativ å stenge av motoren når man er i lufta. I denne oppgaven er sikker tilstand representert ved at både det grønne og det røde LED-lyset skrues av.

Det er ønskelig at styringsenheten skal gå i sikker tilstand om det oppdages alvorlige feil, men enkelte feil kan tolereres til ett visst punkt. Det kan lett oppstå en enkeltstående bitfeil under overføringen, og det er ikke ønskelig å gå i sikker tilstand hver gang dette skjer. Mange bitfeil i løpet av et kort tidsrom er derimot en sterk indikasjon på en noe mer alvorlig feil. For å løse dette problemet benyttes det en feilteller som inkrementeres hver gang det oppdages feil, f.eks. i CRC-koden eller sekvensnummeret som mottas. Om styringsenheten har oppdaget for mange feil vil den gå i sikker tilstand og bryte kommunikasjonen med PCS. Om det er PCS som har oppdaget for mange feil, vil den bryte kommunikasjonen og informere brukeren. Styringsenheten vil så detektere at kommunikasjonen er brutt, og gå i sikker tilstand. I begge tilfeller vil det så bli gjort forsøk på å gjenopprette kommunikasjonen. Feiltelleren vil tilbakestilles til null. Den samme prosedyren for å gjenopprette kommunikasjon vil også benyttes om kommunikasjonen blir brutt på andre måter, f.eks. ved at styringsenheten eller PCS rammes av strømbrudd.

Antallet feil systemet tillater før det aktiverer sikker tilstand kan justeres etter hvor sensitivt for feil man ønsker at systemet skal være. Om det er spesielt stor risiko eller betydelige økonomiske tap knyttet til å aktivere PAS er det ønskelig at systemet skal tolerere flere feil. Det er også mulig å dekrementere feiltelleren hver gang det har gått en viss tid eller er blitt sendt et visst antall meldinger. Om det f.eks. er sannsynlig at det vil være en feil i 0.1 % av meldingene som sendes, kan feiltelleren dekrementeres etter 1000 meldinger. Da vil systemet kun aktivere sikker tilstand om det skjer flere feil innenfor et kort tidsrom, men ignorere sporadiske enkeltfeil.

6.5 TCP

For tilkoblingen mellom styringsenheten og PCS er det benyttet TCP (Transmission Control Protocol), en høynivå kommunikasjonsprotokoll for kommunikasjon over internett. TCP har flere funksjoner som gjør overføringen mer pålitelig, der det viktigste er funksjonalitet for å garantere at meldinger kommer frem, og for å hindre opphopning av meldinger på nettverket (En. congestion control). Dette er viktige funksjoner for sikkerhetskritisk kommunikasjon, men vil ha noen bakdeler. TCP oppretter en fast tilkobling mellom de kommuniserende enhetene ved bruk av et 3-veis “handshake”. Om en av enhetene bryter tilkoblingen utilsiktet, f. eks. ved et strømbrudd, kan det være utfordrende å detektere en brutt tilkobling. Motstykket til TCP, UDP (User Datagram Protocol) har ingen slike garantier. Ved bruk av UDP vil en melding bli sendt uten noen garanti for at den kommer frem. Det kreves derfor tilleggsfunksjoner, som bekreftelser fra mottager om at meldingen kom frem.

Kapittel 7

Implementert Kryptering

Kryptering er viktig for å hindre utenforstående å få tak i sensitiv informasjon som sendes over internett. Kryptering gjør det også vanskelig for angripere å injisere egne kommando-er inn i et system. I kontrollsystemer kan avansert kryptering by på problemer siden det kan gjøre systemet tregere. Det er viktig å undersøke systemkrav og ytelse før kryptering implementeres for å unngå dette.

7.1 Algoritme

Det finnes et stort antall krypteringsalgoritmer, hver tilpasset sitt område. To av de mest brukte algoritmene per i dag er AES (Advanced Encryption Standard) og DES-3 (Data Encryption Standard), som begge er en type blokk-kryptering, som beskrevet i seksjon 4.3. Begge er fremdeles godkjente standarder fra NIST (National Institute of Standards and Technology) [10] [12].

Som et eksempel på mye brukte algoritmer som nå er utdatert, nevnes DES-1 og RC4 [4]. Det ble oppdaget at nøkkellengden på 56 bit og kun en runde kryptering ikke var tilstrekkelig til å hindre brute-force angrep med nyere, raskere datamaskiner. Den tidligere nevnte DES-3 bygger på samme prinsipper som DES-1, men har flere "runder" i krypteringen, altså at teksten blir endret flere ganger utifra krypteringsnøkkelen som er gitt. Den har også en lengre krypteringsnøkkel.

AES er hittil ikke blitt kompromittert. Det finnes teoretiske angrep som muligens vil bli et problem i fremtiden, men som per i dag krever urealistiske mengder regnekraft og lagringskapasitet. I praksis vil de derfor ikke ha noen innvirkning på bruken av AES. [3]

AES ble valgt som krypteringsalgoritme i denne oppgaven, ettersom det eksisterer et offisielt bibliotek for Arduino som bl.a. benytter AES. Det finnes også flere raskere algoritmer spesielt tilpasset Arduino, men disse ble oversett ettersom de har begrenset kompatibilitet med andre systemer.

7.2 Operasjonsmodus

Det finnes flere operasjonsmodus for AES algoritmen. Valg av operasjonsmodus er en viktig del av AES, ettersom flere av dem nå ikke regnes som sikre lenger. De forskjellige modiene kan deles i to hovedgrupper: med og uten autentisering. Om autentisering ikke er en del av operasjonsmodusen som benyttes, må det implementeres i tillegg. Uten autentisering vil det være mulig å garantere konfidensialitet, men ikke integritet. Med andre ord vil innholdet i en melding være konfidensielt, men det vil ikke være mulig å garantere at meldingen ikke har blitt endret på veien, enten med eller uten intensjon. Om autentisering skal implementeres separat blir systemet noe komplekst, og potensialet for feil blir stort. Det er derfor i de fleste tilfeller lønnsomt å benytte operasjonsmodi som kombinerer både konfidensialitet og integritet i en pakke. Riktig valg av operasjonsmodus vil også føre til at den krypterte teksten er forskjellig hver gang, selv om originalteksten før kryptering er den samme. Dette oppnås ved å inkludere et tilfeldig element som brukes som initialiserings-variabel for krypteringsblokken. Uten det tilfeldige elementet, vil det ikke være mulig å lese meldingene direkte, men en som lytter etter meldinger vil likevel kunne se mønster i meldingene som blir sendt, hva som sendes oftest, og hva som sendes hvor for å nevne noe.

I denne oppgaven er det benyttet EAX [9], en operasjonsmodus som inkluderer autentisering og et tilfeldig element. En annen modus som tilfredsstiller disse kravene og som mest sannsynlig er raskere er OCB (Offset CodeBook Mode), men denne har patentbegrensninger. En oversikt over de mest brukte operasjonsmodi og deres egenskaper finnes i tabell 7.1. I denne tabellen ser vi at modusen GCM (Galois/Counter Mode) også har de nødvendige egenskapene. GCM er regnet som et utskiftbart alternativ til EAX og kan i de fleste tilfeller byttes ut med EAX uten at annen kode må endres. EAX ble valgt istedenfor GCM siden det fantes lett tilgjengelige implementasjoner av EAX både for Python og Arduino.

EAX er en effektiv operasjonsmodus, da den legger til minimalt med elementer i meldingene den krypterer. Det vil, ettersom den her benyttes sammen med AES, kun være nødven-

	OCB	EAX	ECB	CBC	GCM	CTR	OFB	CCM
Autentisering	Ja	Ja	Nei	Nei	Ja	Nei	Nei	Ja
Patenter	Ja	Nei	Nei	Nei	Nei	Nei	Nei	Nei
Tilfeldig element	Ja	Ja	Nei	Ja	Ja	Ja	Ja	Ja

Tabell 7.1: Oversikt over hvilke av de mest brukte operasjonsmodiene som har de nødvendige egenskapene.

dig å legge ved MAC-koden og IV. I motsetning til mange andre operasjonsmodus, kan EAX brukes til å kryptere meldinger av ulik lengde, og er derfor ikke avhengig av padding. EAX ble laget for erstatte operasjonsmodusen CCM, som av mange ble sett på som unødvendig kompleks og ineffektiv. CCM er avhengig av padding.

7.3 Nøkkeldistribuering

Symmetrisk

Nøkkeldistribusjon er som tidligere nevnt en av de mest utfordrende emnene innen kryptografi. I tradisjonell symmetrisk kryptering, der begge parter bruker den samme nøkkelen, må det gjøres på en av følgende måter:

- Nøkkelen må sendes over en eksisterende krypteringskanal.
- En utenforstående part som begge de kommuniserende partene stoler på må distribuere nøklene.
- De to kommuniserende partene må begge være i besittelse av nøkkelen fra starten.

Det første alternativet gir lite mening, ettersom man for å bruke en kryptert kanal må opprette den først, og da har man det samme problemet. Alternativ nummer to er mye brukt i kommunikasjon over offentlig internett, men i tilfellet med kontrollsystemer ønsker man å skille kontrollsystemet fra omverdenen i så stor grad som mulig. Det er derfor ikke hensiktsmessig å kommunisere med og stole på en enhet utenfor det lokale nettverket. Det som gjenstår er da at begge enhetene er i besittelse av nøkkelen til å begynne med. Dette er den enkleste metoden, men den krever mer jobb av en eventuell installatør som må sørge for at nøklene stemmer før igangkjøring av systemet, og kan gjøre det vanskeligere å legge til nye enheter i systemet på et senere tidspunkt. For å bytte nøkler er man da også avhengig av å

sende nøklene over den allerede eksisterende kanalen. Dette fører til et ganske åpenlyst problem, nemlig det at en person som har "knekt" krypteringen og fått tilgang til en nøkkel vil ha tilgang til all fremtidig informasjon som blir sendt.

Asymmetrisk

En helt annen fremgangsmåte er å bruke asymmetrisk kryptering for distribusjon av krypteringsnøkler. I asymmetrisk kryptering har ikke lenger de to kommuniserende partene den samme nøkkelen. Istedet har begge parter to nøkler hver, en offentlig og en privat. De er konstruert slik at meldinger kryptert med den offentlige nøkkelen kun kan dekrypteres ved bruk av den private. Dersom en part A vil sende en melding til part B, vil A først be om part B sin offentlige nøkkel og bruke denne til å kryptere meldingen. B vil så motta meldingen og dekryptere denne med sin private nøkkel.

Den mest utfordrende med asymmetrisk kryptering er generering av offentlige og private nøkler. Det viktigste kravet som må stilles til slike nøkler er at det må være umulig å bruke den offentlige nøkkelen til å finne den private. Dette er en selvfølge, ettersom den offentlige nøkkelen vil være lett tilgjengelig, men er utfordrende å gjennomføre i praksis.

RSA

RSA (oppkalt etter utviklerne Ron Rivest, Adi Shamir, og Leonard Adleman) er et mye brukt kryptografisystem som bygger på asymmetriske nøkler. RSA er en relativt treg kryptering, og er derfor sjelden benyttet alene. Den er derimot ofte benyttet for å distribuere nøkler i systemer som benytter symmetrisk kryptering. Da vil en part A generere en nøkkel og sende den til en part B kryptert med B sin offentlige nøkkel. B vil så dekryptere meldingen fra A med sin private nøkkel, og bruke den mottatte symmetriske nøkkelen fra A til videre kryptering. Den offentlige nøkkelen er produktet av to store primtall og et annet tall, og det skal ikke være mulig å dekryptere meldinger som er kryptert med denne nøkkelen uten å kjenne til de originale tallene. For at RSA skal være sikkert, kreves det at de hemmelige primtallene som brukes er *helt tilfeldige*. Hvis ikke vil det i mange tilfeller være mulig å bruke den offentlige nøkkelen til å finne den private. I senere tid har det kommet frem at mange implementasjoner av RSA er usikre, ettersom tallene brukt for å generere nøklene kun er *semi-tilfeldige*. [8]

Figur 7.1: Mann i midten angrep med offentlig-nøkkel kryptering

Nøkkellengden brukt i en RSA kryptering har også innvirkning på hvor sikker krypteringen er. RSA med nøkkellengde 512 bit (64 byte) regnes per i dag som knekt. Den lengste nøkkelen som er faktorisert per februar 2010 (altså hvor man har lyktes i å hente ut de hemmelige tallene fra den offentlige nøkkelen) er 768 bit (96 byte) og dette tok over to år. [7] Basert på tidligere utvikling er det forventet at regnekraft og lagringskapasitet vil bli såpass forbedret at det vil være mulig å faktorisere 1024 bits nøkler innen 2020. Flere har begynt å gå over til å bruke 4096 bit nøkler (512 byte). Til sammenligning har en melding som sendes med ProfiSafe protokollen en maksimal størrelse på 1952 bit (244 byte) 3.2. Dette gjør at de offentlige nøklene må sendes oppdelt i flere meldinger, noe som igjen øker sannsynligheten for feil. Om flere meldinger må brukes til å oppdatere nøkler underveis, vil det også føre til et lengre opphold i oppdateringer fra styringsenhetene. Det må derfor undersøkes om et slikt opphold er forsvarlig, eller gjøres tiltak for å kombinere en del av nøkkelen som skal sendes med data fra styringsenheten.

For å garantere at RSA er sikkert, må det også inkluderes en form for avsenderverifikasjon. Uten dette er krypteringen spesielt utsatt for et såkalt "Mann i midten" angrep, som beskrevet i seksjon 2.2. Partene A og B vil tro de kommuniserer med hverandre, men vil i virkeligheten begge kommunisere med en tredje part som kontrollerer meldingene som blir sendt mellom A og B. Dette er illustrert i figur 7.1.

RSA var planlagt implementert som nøkkeldistribusjon i denne oppgaven. Det ble imidlertid oppdaget at implementasjon av RSA krevde betydelig omstrukturering av den eksisterende koden, og grunnet tidsbegrensninger ble dette ikke prioritert. Istedet er det benyttet en fast nøkkel som begge parter er i besittelse av ved oppstart.

Kapittel 8

Oppsummering og anbefalinger for videre arbeid

8.1 Foreslåtte endringer

Programkoden for PCS ble skrevet i programmeringsspråket Python, versjon 3.5.1. Dette er et høynivå språk. Noen av fordelene ved det er at det er raskere å programmere, det finnes flere innebygde funksjoner, og det er forholdsvis enkelt å lære nye funksjoner. Arduino benytter derimot et språk basert på C, og Arduino kode-biblioteker er skrevet i programmeringsspråket C++, som begge er mer lavnivå språk. Det var derfor utfordrende å skrive kode for kommunikasjon mellom de to språkene. Om det hadde blitt benyttet samme språk i PCS, er det sannsynlig at flere kompatibilitetsproblemer ville vært unngått.

Det benyttes TCP som overordnet kommunikasjonsprotokoll. Dette gir utfordringer under bortfall av en av de kommuniserende partene. Meldingsoppsettet er basert på den lavnivå protokollen ProfiSafe, og denne er bygget på meldinger som sendes uten verifisering og uten fast tilkobling. Dette er mer i tråd med den overordnede protokollen UDP. En av de største fordelene ved bruk av TCP er resending av meldinger som ikke kommer frem intakte. Det er i seksjon 6.3 argumentert for at dette ikke er nødvendig. Alle andre funksjoner i protokollen vil kunne implementeres ved bruk av UDP, og det ser ut til at UDP burde vært valgt over TCP.

Programkoden for Arduino-en kunne med fordel ha vært mer modulær. Det ble gjort forsøk på å skille ut enkelte funksjoner og plassere dem i dedikerte kode-biblioteker, men måten koden benyttet flere funksjoner for å sette sammen en melding på gjorde dette veldig utford-

rende. Som tidligere nevnt, er modularitet et viktig element i kontrollsystemer.

Kommunikasjonen består av meldinger med en fast lengde. Dette er upraktisk, da det ikke åpner for sending av ulike meldinger. Om det skal implementeres en metode for oppdatering av krypteringsnøkler er det behov for lengre meldinger. Ved eventuell bruk av UDP, vil det også være nødvendig å kunne sende egne verifiserings-meldinger. Meldinger av varierende lengde kan implementeres forholdsvis enkelt. Lengden av meldingen legges til som den første byte-en i meldingen. Så leser mottager en byte fra bufferen, sjekker den, og leser inn tilsvarende mengde byte. Det er viktig at lengden på meldingen ikke er kryptert, siden det da ikke vil være mulig å lese lengden før man har tatt inn hele meldingen og dekryptert den.

8.2 Anbefalinger for videre arbeid

Grunnet tidsbegrensninger er det flere områder av dette tema-et det kan jobbes videre med. Noen forslag til videre arbeid følger her.

8.2.1 Nøkkeldistribuering

Det var planlagt å implementere funksjonalitet for asymmetrisk distribuering av krypteringsnøkler, som spesifisert i seksjon 7.3. Det ble imidlertid oppdaget at dette krevde betydelig omstrukturering av den eksisterende koden. Det ble derfor ikke prioritert. Periodisk utskifting av krypteringsnøkler er nødvendig, slik at en angriper som lykkes i å finne en krypteringsnøkkel ikke får tilgang til all fremtidig informasjon.

8.2.2 Videre testing

Det er gjort tester for å verifisere at alle funksjonene som skal detektere feil fungerer slik det er spesifisert. Dette gjelder sekvensnummer, CRC-kode, MAC-kode og feilmeldinger mellom enhetene. Det er testet at nedstenging og gjenoppretting av kommunikasjonskanalen fungerer slik det er spesifisert i seksjon 6.4. Det er også bekreftet at krypteringsalgoritmen fungerer slik den skal. Det er derimot ikke blitt gjort noen forsøk på å kompromittere systemet med ulike nettverksangrep. Sikkerheten i krypteringsalgoritmene som er benyttet er kun teoretisk. For å verifisere at systemet virker, bør det gjøres forsøk på å kompromittere systemet

ved bruk av flere av de vanligste formene for nettverksangrep. Et rent brute force angrep bør forsøkes først.

8.2.3 Dos angrep

Systemet benytter TCP, som har en kjent svakhet mot DoS angrep. TCP krever et 3-trinns handshake for tilkobling, der den første meldingen er en forespørsel om tilkobling. En angriper kan utføre et angrep populært kalt en "SYN flood" (SYN er navnet på en forespørsel om synkronisering/tilkobling) hvor en enhet overøses av tilkoblings-forespørsler og blir ute av stand til å utføre andre oppgaver. I første omgang er det ønskelig at systemet skal kunne detektere et slikt angrep, for så å gå til sikker tilstand. Dette vil kun kreve at en enhet logger antallet tilkoblings-forespørsler og går til sikker tilstand hvis det blir oppdaget unormalt mange av dem, eller forespørsler fra ukjente IP-adresser. En mer utfordrende oppgave vil så være å klare å stenge ned et DoS angrep og fortsette operasjonen som normalt.

Figurer

Tabeller

Tillegg A

Spesifikasjoner

A.1 Arduino Mega 2560

Tekniske spesifikasjoner for Arduino Mega 2560

Microkontroller	ATmega2560
Spenning	5V
Input spenning (anbefalt)	7-12V
Input spenning (grense)	6-20V
Digitale I/O pinner	54
Analoge Input pinner	16
DC strøm per I/O pinne	20 mA
DC strøm for 3.3V pinne	50 mA
Flash minne	256 KB (8 KB brukt til bootloader)
SRAM	8 KB
EEPROM	4 KB
Klokke frekvens	16 MHz
Lengde	101.52 mm
Bredde	53.3 mm
Vekt	37 g

A.2 Arduino Ethernet Shield

Tekniske spesifikasjoner for Arduino Ethernet Shield

Spenning	5V
Ethernet kontroller	W5100 med egen buffer på 16K
Hastighet	10/100Mb

Tilkobling til Arduino på SPI port

Tillegg B

Innhold på CD

En CD er veldagt på slutten av denne oppgaven. Den inneholder følgende:

- Rapporten i PDF-format
- Kildekoden til rapporten, hentet fra LaTeX
- UML og annen dokumentasjon av veldagt kode
- Arduino kode
- Python kode
- Ulike kodebiblioteker benyttet i prosjektet

Bibliografi

[1] Barthel, H. (1999). ProfiSafe, Profile for Failsafe Technology, V1.0. *PROFIBUS DP/PA*.

[2] Bellare, M. and Namprempre, C. (2007). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology, ASIACRYPT 2000*, 1976:531–545.

[3] Bogdanov, A., Khovratovich, D., and Rechberger, C. (2011). Biclique cryptanalysis of the full AES. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 344–371.

[4] Bruen, A. A. and Forcinito, M. A. (2005). *Cryptography, Information Theory and Error-correction: A handbook for the 21st century*. Wiley, Hoboken, NJ, 1st edition.

[5] Eriksen, H. B. (Juni 2013). Struktur og sikkerhet av nettverk ved integrerte operasjoner. *Masteroppgave NTNU*.

[6] Grøtan, T. O., Jaatun, M. G., Øien, K., and Onshus, T. (2007). The SeSa method for assessing secure remote access to safety instrumented systems. *Sintef rapport*.

[7] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A., Thomé, E., Bos, J., Gaudry, P., Kruppa, A., Montgomery, P., Osvik, D. A., te Riele, H., Timofeev, A., and Zimmermann, P. (2010). Factorization of a 768-bit RSA modulus. Cryptology ePrint Archive, Report 2010/006. http://eprint.iacr.org/.

[8] Lenstra, A. K., Hughes, J. P., Augier, M., Bos, J. W., Kleinjung, T., and Wachter, C. (2012). Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064. http://eprint.iacr.org/.

[9] Mihir Bellare, Phillip Rogaway, D. W. (2004). The EAX Mode of Operation (A two-pass authenticated encryption scheme optimized for simplicity and efficiency). *Fast Software Encryption (FSE), LNCS*, 3017:389–407.

[10] National Institute of Standards and Technology (2001). Announcing the advanced encryption standard. *Processing Standards Publication*, 197.

[11] The Institute of Electrical and Electronics Engineers, Inc. (2012). IEEE Standard for Ethernet. *IEEE Std 802.3™*.

[12] William C. Barker, E. B. (2012). Recommendation for the triple data encryption algorithm (TDEA) block cipher. *NIST Special Publication 800-67*.